Highly Parallel, High-Precision Numerical Integration

David H. Bailey[1] and Sinai Robins[2]

Abstract

In this paper, we investigate the use of modern parallel computer technology to obtain high-precision numerical values for ordinary and iterated integrals. Such numerical values can be used to experimentally find analytic values. In a previous paper, three schemes were described for high-precision one-dimensional integration. Two of these three were able to evaluate 14 difficult test problems, including problems with vertical derivatives, blow-up singularities and infinite intervals, to 1000-digit accuracy.

This paper describes the implementation of one of these schemes on a highly parallel computer system. On 1024 processors our program achieves a speedup of 807 times, yet on each problem produces results to nearly 2000-digit accuracy. This paper also discusses two-dimensional numerical integration, which is much more expensive because many more function evaluations must be performed. Nonetheless, on a test suite of eight rigorous two-dimensional problems, our programs achieve nearly 100-digit accuracy, with a parallel speedup of 627 times on 1024 processors.

Keywords: numerical quadrature, numerical integration, arbitrary precision

[1]Lawrence Berkeley National Laboratory, Berkeley, CA 94720 dhbailey@lbl.gov. This work was supported in part by the Director, Office of Computational and Technology Research, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy, under contract number DE-AC03-76SF00098. This work was also supported in part by the National Science Foundation, under Grant DMS-0342255.

[2]Department of Mathematics, Temple University, Philadelphia, PA 19122 srobins@math.temple.edu.

1. Introduction

The high-precision evaluation of numerical integrals is emerging as a useful tool for experimental mathematics. In many cases, the numerical value, if computed to sufficiently high precision and combined with integer relation detection schemes such as PSLQ [4], can be used to discover the analytic evaluation (i.e., a closed-form formula) for the integral. The precision required in these computations is typically 100-digit accuracy or so, but often multi-hundred or even multi-thousand-digit accuracy is required in order for the integer relation detection schemes to obtain a numerically meaningful result.

As one example, recently one of the present authors, together with Jonathan Borwein and Greg Fee of Simon Fraser University in Canada, were inspired by a recent problem in the *American Mathematical Monthly* [1]. They found by using an earlier single-processor version of the quadrature scheme described in this paper, together with a PSLQ integer relation detection program, that if $C(a)$ is defined by

$$C(a) \;=\; \int_0^1 \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)},$$

then

$$\begin{aligned} C(0) &= \pi\log 2/8 + G/2 \\ C(1) &= \pi/4 - \pi\sqrt{2}/2 + 3\sqrt{2}\arctan(\sqrt{2})/2 \\ C(\sqrt{2}) &= 5\pi^2/96. \end{aligned}$$

Here $G = \sum_{k\ge 0}(-1)^k/(2k+1)^2$ is Catalan's constant. These experimental results then led to the following general result, rigorously established, among others:

$$\int_0^\infty \frac{\arctan(\sqrt{x^2+a^2})\,dx}{\sqrt{x^2+a^2}(x^2+1)} \;=\; \frac{\pi}{2\sqrt{a^2-1}}\left[2\arctan(\sqrt{a^2-1}) - \arctan(\sqrt{a^4-1})\right].$$

As a second example, recently Jonathan Borwein and one of the present authors empirically determined that

$$\begin{aligned} \frac{2}{\sqrt{3}}\int_0^1 \frac{\log^6(x)\arctan[x\sqrt{3}/(x-2)]}{x+1}\,dx \;=\; \frac{1}{81648}\,[&-229635L_3(8) \\ &+29852550L_3(7)\log 3 - 1632960L_3(6)\pi^2 + 27760320L_3(5)\zeta(3) \\ &-275184L_3(4)\pi^4 + 36288000L_3(3)\zeta(5) - 30008L_3(2)\pi^6 \\ &-57030120L_3(1)\zeta(7)], \end{aligned}$$

where $L_3(s) = \sum_{n=1}^\infty [1/(3n-2)^s - 1/(3n-1)^s]$. Based on these experimental results, general results of this type have been conjectured but not yet rigorously established.

The above examples are ordinary one-dimensional integrals. Two-dimensional integrals are also of interest. Along this line, recently Jonathan Borwein and one of the present authors determined that

$$\frac{2}{3}\int_0^1\int_0^1 \sqrt{x^2+y^2}\,dx\,dy + \frac{1}{3}\int_0^1\int_0^1 \sqrt{1+(u-v)^2}\,du\,dv$$
$$= \frac{1}{9}\sqrt{2} + \frac{5}{9}\log(\sqrt{2}+1) + \frac{2}{9}.$$

See [3] for further details and some additional examples.

2. The Tanh-Sinh Quadrature Algorithm

In a previous paper [6], one of the present authors and his co-authors described three high-precision one-dimensional quadrature schemes, and exhibited results for computer runs with 400-digit and to 1000-digit precision. The three schemes were: (1) Gaussian quadrature, (2) error function quadrature, and (3) tanh-sinh quadrature. The authors concluded that of these three schemes, the tanh-sinh scheme holds the best promise for very high-precision usage. It features an initialization procedure that is fundamentally faster than the other two schemes, the ability to obtain fully accurate results even for many integrand functions with vertical derivatives or blow-up singularities at endpoints and generally very fast run times.

Both the tanh-sinh scheme and the error function scheme are based on the Euler-Maclaurin summation formula, which implies that for certain bell-shaped integrands, a simple block-function approximation to the integral is remarkably accurate, much more so that one would normally expect [2, pg. 180]. This principle is utilized by transforming an integral of some function $f(x)$ on the interval $[-1,1]$ to an integral on $(-\infty,\infty)$ using the change of variable $x = g(t)$. Here $g(x)$ is some monotonic, infinitely differentiable function with the property that $g(x) \to 1$ as $x \to \infty$ and $g(x) \to -1$ as $x \to -\infty$, and also with the property that $g'(x)$ and all higher derivatives rapidly approach zero for large positive and negative arguments. In this case one can write, for $h > 0$,

$$\int_{-1}^{1} f(x)\,dx \;=\; \int_{-\infty}^{\infty} f(g(t))g'(t)\,dt \;=\; h\sum_{j=-\infty}^{\infty} w_j f(x_j) + E(h),$$

where $x_j = g(hj)$ and $w_j = g'(hj)$. If $g'(t)$ and its derivatives tend to zero sufficiently rapidly for large t, positive and negative, then even in cases where $f(x)$ has a vertical derivative or an integrable singularity at one or both endpoints, the resulting integrand $f(g(t))g'(t)$ will be a smooth bell-shaped function for which the Euler-Maclaurin argument applies. In these cases, the Euler-Maclaurin formula implies that the error $E(h)$ in this approximation decreases faster than any power of h. Indeed, as we shall see, tanh-sinh quadrature often achieves quadratic convergence, namely a doubling of the number of correct digits in the result when the interval h is reduced by half.

The tanh-sinh scheme uses $g(t) = \tanh(\pi/2 \cdot \sinh t)$. It approximates the integral of $f(t)$ on $[-1,1]$ by the sum $\sum_{j=-N}^{N} w_j f(x_j)$, where the abscissas are given by $x_j = g(hj)$ and the weights are given by $w_j = g'(hj)$. In our implementation, the parameter h is set to 2^{-k}, where k is the "level" of the quadrature calculation. Successively larger levels reduce h in half, double the number of abscissa-weight pairs (and thus double the number of

function evaluations required in a quadrature calculation), but also approximately double the number of correct digits in the result, in many cases, as mentioned above. In our implementation, the parameter N is chosen large enough so that $w_j < \epsilon_2$ for all $j > N$, where $\epsilon_2 = \epsilon_1^2$, $\epsilon_1 = 10^{-p_1}$, and p_1 is the primary precision target accuracy (2000 digits in this case). These extra-small weights, much smaller than the primary epsilon, permit one to obtain accurate results for many functions with blow-up singularities at endpoints. The abscissa-weight pairs are pre-computed and stored for multiple integration calculations. Full details are given in [6].

The tanh-sinh integration scheme was first introduced by Takahasi and Mori [10].

3. High-Precision Arithmetic

The Arbitrary Precision (ARPREC) computation library was used to perform the required high-precision arithmetic computations described in this paper [5]. This software library is written in C++, but it includes both C++ and Fortran-90 translation modules, so that existing C++ and Fortran-90 application programs can utilize this library by making only very minor changes to the source code. In most cases, it is only necessary to change type statements and input/output statements of the variables that one wishes to be treated as arbitrary precision, and all other operations are automatically performed by the library. One fortunate feature of high-precision numerical quadrature, as described in this paper, is that all individual high-precision arithmetic operations and transcendental function evaluations can be performed locally on a single processor. Thus it is not necessary to invoke parallel processing within the ARPREC library itself for the test problems considered below.

4. 1-D Parallel Implementation

Among its virtues, the tanh-sinh quadrature scheme is well suited for implementation on a highly parallel computer system. This is because both the initialization procedure (the generation of abscissa-weight pairs) and the evaluation of the integrand function at these abscissas are inherently parallel operations—each instance theoretically can be done independently, although there are numerous details that must be observed to avoid serious reductions in parallel performance.

Parallel processing was invoked in the computations described below using the Message Passing Interface (MPI) library [7]. Calls to the MPI library were used for synchronization, collection of distributed data to a single processor, and the broadcast of data residing on a single processor to all other processors. The parallel system we used is organized into 16-processor shared memory nodes, but no attempt was made to exploit shared-memory parallelism—instead each node contained 16 tasks of our MPI program.

One straightforward way to perform the tanh-sinh scheme in parallel is as follows: (1) on each processor, zero out an array of arbitrary precision values that is large enough to hold all anticipated abscissa-weight pairs; (2) distribute the computation of abscissa-weight pairs using a cyclic scheme, where the abscissa-weight pair indexed j, namely (x_j, w_j), is assigned to and computed on processor $p = j \bmod n$, where n is the number of processors; (3) combine the abscissa-weight arrays on individual processors into a single

array that has all contributions, by means of a global, element-wise high-precision sum reduction operation, which is then broadcast to all processors; (4) perform the function evaluations, using some scheme (such as a cyclic scheme) to evenly distribute the workload among processors; (5) sum the function-weight products on individual processors to a single high-precision value, then send these to one processor, where they are added together, with the final sum broadcast to all processors. This scheme appears to achieve good scalability up to about 100 processors.

It is possible to achieve even higher scalability by modifying the above scheme as follows: (1) calculate, in parallel, the abscissa-weight pairs, but do *not* distribute any of the pairs calculated on an individual processor to any other processor; (2) during the quadrature calculation, perform on an individual processor only those function evaluations associated with its particular set of abscissa-weight pairs.

One difficulty with this second approach derives from the fact, as we shall see in the problems studied below, that different integration problems require different numbers of abscissa-weight pairs to achieve a given accuracy level. Even among the problems described in the next section, some achieve full 2000-digit accuracy with only 9 levels of abscissa-weight pairs (corresponding to $h = 2^{-9}$), while others require 13 levels of abscissa-weight pairs (corresponding to $h = 2^{-13}$), or in other words 16 times as many pairs and thus 16 times as many function evaluations. Using all 13 levels for all problems is clearly rather wasteful. Note that if one has computed 13 levels of abscissa-weight pairs, one can compute with only nine levels, for instance, by accessing the abscissa-weight array with a stride of 16.

Unfortunately, if a straightforward cyclic scheme is used to assign the abscissa-weight pairs to processors, a power-of-two number of processors is used (the most common and convenient case in parallel computing), and the abscissa-weight array is accessed with a power-of-two stride, as mentioned in the paragraph above, then a catastrophic load imbalance ensues—some processors have a large fraction of the pairs (and corresponding function evaluations), while other processors have none at all.

It turns out not to be easy to find an efficient assignment scheme for this application. Several popular "hashing" schemes from the computer science literature were found to be ineffective. Even a "random" scheme, or in other words a scheme that employs a good pseudo-random number generator, gives rather disappointing results. Table 1 gives the results of some tests of five different assignment schemes, among numerous ones that we tried. In these tests, 70,000 indices (the approximate number of abscissa-weight pairs actually generated in the quadrature computations described in the Sections 5 and 6) were assigned to processors according to the five schemes. The smallest and largest number of indices assigned to any processor by a given scheme, for various processor numbers and strides, are then shown in the table. The more nearly equal these max and min figures are, the better the assignment scheme. The five assignment schemes are:

1. CYC, a cyclic scheme: $p = \mathrm{mod}(j, n)$
2. BC1, a block-cyclic scheme: $p = \mathrm{mod}(\lfloor j/16 \rfloor, n)$

		CYC		BC1		BC2		MCBC		RAND	
Proc.	Stride	Min	Max	Min	Max	Min	Max	Min	Max	Min	Max
4	1	17500	17500	17489	17504	17493	17509	17499	17501	117426	17578
4	2	0	17500	8744	8752	8232	9267	8748	8752	8655	8828
4	4	0	17500	4372	4376	4116	5148	4372	4376	4355	4397
4	8	0	8750	2186	2188	2058	2574	2186	2188	2166	2233
4	16	0	4375	1093	1094	1029	1287	1093	1094	1047	1148
16	1	4375	4375	4368	4384	4369	4386	4374	4376	4267	4505
16	2	0	4375	2184	2192	2056	2322	2187	2188	2133	2247
16	4	0	4375	1092	1096	1028	1290	1093	1094	1037	1135
16	8	0	4375	546	548	514	774	546	547	500	596
16	16	0	4375	273	274	257	516	273	274	246	305
64	1	1093	1094	1088	1104	1088	1105	1093	1095	1022	1169
64	2	0	1094	544	552	512	585	545	548	505	600
64	4	0	1094	272	276	256	325	272	276	242	315
64	8	0	1094	136	138	128	195	136	138	112	176
64	16	0	1094	68	69	64	130	68	69	48	87
256	1	273	274	272	288	272	289	272	274	234	320
256	2	0	274	136	144	128	153	136	137	106	168
256	4	0	274	68	72	64	85	68	69	45	93
256	8	0	274	34	36	32	51	34	35	17	51
256	16	0	274	17	18	16	34	17	18	6	29
1024	1	68	69	64	80	68	85	67	69	42	97
1024	2	0	69	32	40	32	45	32	36	16	54
1024	4	0	69	16	20	16	25	16	20	6	30
1024	8	0	69	8	10	8	15	8	10	0	19
1024	16	0	69	4	5	4	10	4	5	0	12

Table 1: Min/max processor counts for five assignment functions (70,000 indices)

3. BC2, a block-cyclic scheme: $p = \mathrm{mod}(\lfloor j/17 \rfloor, n)$

4. MCBC, a mixed cyclic, block-cyclic scheme: $p = \mathrm{mod}(j + \lfloor j/16 \rfloor, n)$

5. RAND, a pseudo-random scheme: $p = \lfloor z_j n \rfloor$, where z_j is a uniform generator on $(0, 1)$.

In the above, "mod" is used to mean the function that returns the remainder when the first argument is divided by the second argument.

It can be seen from the results in Table 1 that of the five schemes mentioned, the one we have named the mixed cyclic, block-cyclic scheme (MCBC) is the best. It provides a virtually perfect load balance across a large range of processors (up to 1024) and strides (up to stride 16). This is the scheme that was used in the computations described below.

5. 1-D Test Problems

The following 14 integrals are taken from the suite used in the earlier paper [6]. They are typical of the integrals that have been encountered in experimental math research,

except that in each of these cases an analytic result is known, as shown below, facilitating the checking of results:

- 1–4: Continuous functions on finite intervals.
- 5–6: Continuous functions on finite intervals, but with a vertical derivative at an endpoint.
- 7–10: Functions on finite intervals with an integrable singularity at an endpoint.
- 11–13: Functions on an infinite interval.
- 14: An oscillatory function on an infinite interval.

$$
\begin{aligned}
1 &: \int_0^1 t\log(1+t)\,dt = 1/4 \\
2 &: \int_0^1 t^2 \arctan t\,dt = (\pi - 2 + 2\log 2)/12 \\
3 &: \int_0^{\pi/2} e^t \cos t\,dt = (e^{\pi/2} - 1)/2 \\
4 &: \int_0^1 \frac{\arctan(\sqrt{2+t^2})}{(1+t^2)\sqrt{2+t^2}}\,dt = 5\pi^2/96 \\
5 &: \int_0^1 \sqrt{t}\log t\,dt = -4/9 \\
6 &: \int_0^1 \sqrt{1-t^2}\,dt = \pi/4 \\
7 &: \int_0^1 \frac{\sqrt{t}}{\sqrt{1-t^2}}dt = 2\sqrt{\pi}\Gamma(3/4)/\Gamma(1/4) \\
8 &: \int_0^1 \log t^2\,dt = 2 \\
9 &: \int_0^{\pi/2} \log(\cos t)\,dt = -\pi\log(2)/2 \\
10 &: \int_0^{\pi/2} \sqrt{\tan t}\,dt = \pi\sqrt{2}/2 \\
11 &: \int_0^{\infty} \frac{1}{1+t^2}\,dt = \pi/2 \\
12 &: \int_0^{\infty} \frac{e^{-t}}{\sqrt{t}}\,dt = \sqrt{\pi} \\
13 &: \int_0^{\infty} e^{-t^2/2}\,dt = \sqrt{\pi/2} \\
14 &: \int_0^{\infty} e^{-t}\cos t\,dt = 1/2
\end{aligned}
$$

Note that the integrals on an infinite interval, which in each case here is $[0,\infty)$, can be transformed to integrals on a finite integral, which is required for tanh-sinh quadrature, by the transformation $s = 1/(t+1)$.

Problem	Levels	Processors					
Number	Required	1	4	16	64	256	1024
Init		22462.17	5668.95	1439.76	360.53	92.79	25.92
1	10	1952.09	499.64	125.95	31.98	8.34	3.19
2	10	5575.73	1433.29	363.06	92.46	24.65	7.85
3	10	2865.02	732.78	186.25	46.51	12.46	4.00
4	10	6220.04	1596.34	403.42	103.33	27.26	8.43
5	9	986.87	254.09	64.00	16.48	4.58	1.42
6	10	105.40	27.21	6.85	1.75	0.48	0.26
7	10	223.78	58.06	14.40	3.76	0.95	0.33
8	9	975.23	249.93	63.94	16.42	4.56	1.42
9	10	3078.12	790.60	201.44	51.32	13.28	4.03
10	10	1377.10	361.05	91.28	23.65	6.09	1.97
11	11	91.37	23.45	6.00	1.55	0.42	0.24
12	12	3305.49	838.17	211.60	53.53	13.82	4.21
13	13	4469.49	1136.02	284.50	71.78	18.55	5.00
14	13	13960.36	3595.45	907.79	231.07	59.12	15.50
Total		67648.26	17265.03	4370.24	1106.12	287.35	83.77
Ratio		1.00	3.92	15.48	61.16	235.42	807.55

Table 2: Parallel run times and speedup ratios for 1-D problems

6. 1-D Performance Results

The results of the 1-D parallel quadrature tests are given in Table 2. The first line gives the run time, in seconds, for the initialization process. The initialization time is listed here separately from the integral evaluations, since it is expected that in many practical applications, the abscissas and weights will be computed once and then stored for numerous subsequent evaluations. In all tests, the full target accuracy of 10^{-2000} was achieved, except in Problem 14, where the accuracy was 10^{-1972}. This lower accuracy was not due to any difficulty of implementation, but, as it turns out, this is the best that can be achieved with 13 levels of abscissas and weights (i.e., with $h = 2^{-13}$, which requires approximately 71,000 abscissa-weight pairs) on this particular problem. When 14 levels (or 13 levels with a slightly smaller initial h) are used, the error is reduced to less than 10^{-2000}, although the run time approximately is correspondingly higher.

These runs were made on the "Seaborg" system, an IBM Power3 parallel supercomputer at the Lawrence Berkeley Laboratory, except that the one-processor and four-processor results were performed on "Hockney," a smaller system that has the same processor design and clock rate. The parallel program performs all functions of the single processor code, including the calculation of an estimated error in the result [6]. The one-processor figures shown in Table 2 are based on an efficient single-processor version of this program, with no calls to the MPI parallel library or other modifications for parallel computing. Thus the speedup ratios shown in the table are true comparisons to

single-processor performance.

It is evident from these results that the parallel quadrature program is achieving very nearly perfect speedup up to several hundred processors. This ratio drops a bit from ideal at 1024 processors, but it is clear from examining these timings (especially for Problems 5 through 11) that this is due in part to the fact that when these problems are divided into 1024 pieces, there is not a lot for each processor to do. Also, in a parallel implementation, a few more evaluations of the function must be performed in a highly parallel environment than on a single-processor system, because it is more difficult to determine when the function-weight products are sufficiently small that the summation may be terminated.

7. Two-Dimensional Quadrature

The tanh-sinh scheme described above can be generalized to two or more dimensions. In particular, a 2-D iterated integral can be approximated as follows:

$$\begin{aligned}\int_{-1}^{1}\int_{-1}^{1} f(x,y)\,dx\,dy &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(g(s),g(t))g'(s)g'(t)\,ds\,dt \\ &= h\sum_{k=-\infty}^{\infty}\sum_{j=-\infty}^{\infty} w_j w_k f(x_j,x_k) + E(h),\end{aligned}$$

where $g(t) = \tanh(\pi/2 \cdot \sinh(t))$ as in the 1-D case, and where x_j and w_j are the 1-D abscissas and weights. This same approach can easily be extended to numerically evaluate more general integrals of the form

$$\int_a^b \int_{c(y)}^{d(y)} f(x,y)\,dx\,dy.$$

As before, the Euler-Maclaurin formula asserts that for a certain class of functions $f(x,y)$, including many with vertical derivatives and blow-up singularities at the boundaries of the rectangle, the error $E(h)$ in the above approximation goes to zero faster than any power of h. As a result, 2-D tanh-sinh quadrature, like the 1-D version, often achieves quadratic convergence, wherein each additional level of abscissa-weight pairs yields twice as many correct digits in the result.

However, 2-D quadrature inherently is much more expensive than 1-D quadrature, because the number of function evaluations in a 2-D array, assuming the same overall spacing, is many times larger than in a 1-D problem. Millions of function evaluations may be required to obtain, say, 100-digit accuracy in the result. Also, we have found that 2-D tanh-sinh scheme is more sensitive to anomalies such as non-differentiable or blow-up singularities at boundaries. In such cases, we have found that each additional level typically yields only about 1.4 times as many correct digits, whereas in 1-D quadrature, problems with similar anomalies typically exhibit quadratic convergence (each additional level approximately doubles the number of correct digits). What's more, in 2-D quadrature, each additional level *quadruples* the computational cost instead of merely doubling the cost, since four times as many function evaluations are required.

Higher-order variants of the Euler-Maclaurin, and their implication for quadrature, are discussed in [9].

8. Implementation of 2-D Quadrature

Our serial implementation of a 2-D tanh-sinh scheme is a reasonably straightforward extension of our 1-D program, although the handling of error estimates must be done quite carefully. On the bright side, in 2-D tanh-sinh quadrature the initial computation of abscissas and weights is a trivial matter, in comparison with the millions of functions evaluations that are required, and so does not dominate the computation as it does in the 1-D case.

Our parallel implementation of the 2-D scheme again relies crucially on a carefully chosen scheme for allocating processors to the abscissa array for function evaluations. Our program assigns a batch of 16 consecutively numbered processors to each column, and then assigns the function evaluations in this column among these 16 processors. In this way, the program exploits available parallelism in both dimensions. The particular assignment scheme used by the program is as follows: the 16 processors p that satisfy $\lfloor p/16 \rfloor = \mod(j+j/16, n/16)$ are assigned to column j of the 2-d array of abscissas. Then within column j, location (i, j) is assigned to the processor p that satisfies $\mod(p, 16) = \mod(i + i/16, 16)$. Note that both rows and columns employ a mixed cyclic, block-cyclic scheme, which provides an even load balance for function evaluations, yet avoids difficulties with power-of-two strides.

9. 2-D Test Problems

We tested our serial and parallel implementations of this 2-D tanh-sinh quadrature scheme on a suite of eight test problems. As before, this set includes some rather difficult examples, including one problem with a non-differentiable point at a boundary (Problem 1), two problems with a blow-up singularity at a boundary (Problems 4 and 6), two problems where the inner integral boundary is not merely an interval but instead bounded by two functions (Problems 7 and 8), and one problem on an infinite interval (Problem 5).

$$1 \;:\; \int_0^1 \int_0^1 \sqrt{s^2+t^2}\, ds\, dt \;=\; \sqrt{2}/3 - \log(2)/6 + \log(2+\sqrt{2})/3$$

$$2 \;:\; \int_0^1 \int_0^1 \sqrt{1+(s-t)^2}\, ds\, dt \;=\; -\sqrt{2}/3 - \log(\sqrt{2}-1)/2 + \log(\sqrt{2}+1)/2 + 2/3$$

$$3 \;:\; \int_{-1}^1 \int_{-1}^1 (1+s^2+t^2)^{-1/2}\, ds\, dt \;=\; 4\log(2+\sqrt{3}) - 2\pi/3$$

$$4 \;:\; \int_0^\pi \int_0^\pi \log[2-\cos s - \cos t]\, ds\, dt \;=\; 4\pi G - \pi^2 \log 2$$

$$5 \;:\; \int_0^\infty \int_0^\infty \sqrt{s^2+st+t^2}\, e^{-s-t}\, ds\, dt \;=\; 1 + 3/4 \cdot \log 3$$

$$6 \;:\; \int_0^1 \int_0^1 (s+t)^{-1}[(1-s)(1-t)]^{-1/2}\, ds\, dt \;=\; 4G$$

$$7 \;:\; \int_0^1 \int_0^t (1+s^2+t^2)^{-1/2}\, ds\, dt \;=\; -\pi/12 - 1/2 \cdot \log 2 + \log(1+\sqrt{3})$$

$$8 \;:\; \int_0^\pi \int_0^t (\cos s \sin t) e^{-s-t}\, ds\, dt \;=\; 1/4 \cdot (1+e^{-\pi})$$

Problem Number	Levels Required	Processors				
		1	16	64	256	1024
1	9	4854.89	384.60	98.76	25.53	10.51
2	6	72.56	5.96	1.70	0.97	2.43
3	7	328.90	26.21	6.91	2.36	3.17
4	9	60475.50	4826.64	1228.59	307.82	82.08
5	9	8973.73	696.87	177.61	45.26	14.93
6	9	6448.27	495.78	127.50	32.78	11.60
7	6	91.13	7.34	2.04	1.04	2.45
8	6	449.40	36.10	9.66	3.08	3.13
Total		81694.38	6479.50	1652.77	418.84	130.30
Ratio		1.00	12.61	49.43	195.05	626.97

Table 3: Parallel run times and speedup ratios for 2-D problems

Because of the much higher computational cost of 2-D quadrature, due to the much larger number of function evaluations required, we set a more modest goal of 100-digit accuracy in these problems. We employed 120-digit working precision.

10. 2-D Performance Results

Performance results for the 2-D quadrature program are shown in Table 2. In each problem we achieved over 100-digit accuracy, except for Problems 4 and 6, where the errors were 10^{-86} and 10^{-80}, respectively. No results are shown in this table for four processors, since the parallel program assumes a minimum of 16 processors.

It is clear from these results that, unlike the 1-D case, there is a large difference in run times between well-behaved integrands and those with singularities at a corner or boundary. Those without such anomalies can be evaluated to over 100-digit accuracy with just six levels, requiring only a few minutes run time. For those problems that do exhibit such anomalies, nine levels are needed, requiring many more function evaluations. Indeed, for Problems 4 and 6, even nine levels of abscissa-weight pairs (and the corresponding 2-D mesh of function evaluations) evidently were not sufficient—it appears that one additional level would be required in each case to achieve over 100 digit accuracy, which would multiply the run times by a factor of four.

The parallel speedups for 2-D quadrature are not as high as for 1-D quadrature, in part because the handling of error estimation is more complicated than in the 1-D case. Also, much of this reduction in scalability was rooted in the shorter-running problems, whose modest computational work evidently cannot be as efficiently distributed among 1024 processors. The speedup factors for problems with nine levels are significantly higher. Note, for instance, that the parallel speedup for Problem 4, the longest running problem, is 737, and for Problems 4, 5 and 6 together is 699.

The speedup figures shown in Table 2 are for a program strategy that employs parallel decomposition in both row and column dimensions, as described in Section 8. We also

tried a scheme that only distributed the columns of the 2-D array of abscissas to be evaluated. This scheme has some advantages, including a greater flexibility for modest levels of parallelism. But it is less effective at the high end, achieving only 508 times speedup for 1024 processors.

11. Conclusions

We have demonstrated serial and parallel implementations of 1-D and 2-D tanh-sinh quadrature schemes. These programs were evaluated using two suites of challenging test problems, including problems with nondifferentiable and blow-up singularities on the boundaries, oscillatory behavior, and problems on infinite intervals. The 1-D program evaluated each of the the 1-D test problems to nearly 2,000-digit precision; the 2-D program evaluated each of the the 2-D test problems to nearly 100-digit precision. The serial programs required 18 and 22 hours, respectively to complete the test suites, but these times were reduced to only 83 seconds and 130 seconds, respectively, by the parallel programs (using 1024 processors). The more modest precision achieved by the 2-D program reflects the dramatically higher computational requirement of 2-D quadrature, which often requires millions of function evaluations to achieve even 100-digit accuracy. Our parallel programs achieved very high scalability figures—807 times overall speedup on a 1024-processor system for the 1-D problem suite, and 627 times overall speedup for the 2-D problem suite.

One can also consider performing 3-D quadrature with this same approach. The challenge here is that once again the number of required function evaluations is multiplied by a large factor. Thus it appears that high-precision (100-digit or higher) 3-D quadrature calculations are not likely to be feasible in reasonable run time in the near future, except for some well-behaved cases (or in cases where one or both of the inner integrals can be evaluated analytically). But 3-D quadrature is possible, using this approach, for ordinary 16-digit IEEE machine precision, or 32-digit "double-double" arithmetic, which can easily be implemented on IEEE-compliant systems [8].

References

[1] Zafar Ahmed, "Definitely an Integral," *American Mathematical Monthly*, vol. 109 (2002), no. 7, pg. 670–671.

[2] Kendall E. Atkinson, *Elementary Numerical Analysis*, John Wiley and Sons, 1993.

[3] David H. Bailey, Jonathan M. Borwein, Vishaal Kapoor and Eric Weisstein, "Ten Problems in Experimental Mathematics: A Challenge," manuscript, 2004, available from the URL http://crd.lbl.gov/~dhbailey/dhbpapers/tenproblems.pdf.

[4] David H. Bailey and David Broadhurst, "Parallel Integer Relation Detection: Techniques and Applications," *Mathematics of Computation*, vol. 70, no. 236 (2000), pg. 1719-1736.

[5] David H. Bailey, Yozo Hida, Xiaoye S. Li and Brandon Thompson, "ARPREC: An Arbitrary Precision Computation Package," technical report LBNL-53651, software and documentation available from the URL http://crdl.bl.gov/~dhbailey/mpdist.

[6] David H. Bailey, Xiaoye S. Li and Karthik Jeyabalan, "A Comparison of Three High-Precision Quadrature Programs," manuscript, available from the URL http://crd.lbl.gov/~dhbailey/dhbpapers/quadrature.pdf.

[7] William Gropp, Ewing Lusk, Anthony Skjellum, *Using MPI: A Portable Parallel Programming with the Message-Passing Interface*, MIT Press, Cambridge, MA, 1996.

[8] Yozo Hida, Xiaoye S. Li and David H. Bailey, "Algorithms for Quad-Double Precision Floating Point Arithmetic," *15th IEEE Symposium on Computer Arithmetic*, IEEE Computer Society, 2001, pg. 155–162.

[9] Yael Karshon, Shlomo Sternberg and Jonathan Weitsman, "The Euler-Maclaurin Formula for Simple Integral Polytopes," *Proceedings of the American Academy of Science*, vol. 100 (2003), no. 2, pg. 426–433.

[10] H. Takahasi and M. Mori, "Double Exponential Formulas for Numerical Integration," Publications of RIMS, Kyoto University, vol. 9 (1974), pg. 721–741.